SUPPLEMENTARY MATERIAL

This material supplements the paper "Extracting Top-k Frequent and Diversified Patterns in Knowledge Graphs". It includes pseudocodes, proofs, complexity analysis, and complementary experimental results for kCP and kCPD.

APPENDIX A
PSEUDOCODES

We first present the pseudocodes of solutions for kCP and kCPD in the following.

Algorithm 1, denoted as kCP-B, summarizes the procedures of the baseline solution to kCP problem (see in Section 3.1). It invokes two sub-routines, i.e., pattern extension (Line 9) and MNI evaluation (Line 6), which correspond to the Algorithm 2 and Algorithm 3, respectively.

Algorithm 1 kCP-B(KG, P, k)

1: Initialize top-k frequent pattern result min-heap $\mathcal{H} \leftarrow \emptyset$
2: Initialize max-heap candidate pattern set $\mathcal{CS} \leftarrow (\mathsf{P}, 0)$
3: **while** $\mathcal{CS}$ is not empty **do**
4: $\quad \mathsf{P}' \leftarrow \mathcal{CS}.\mathsf{top}(); \mathcal{CS}.\mathsf{delete\_max}()$
5: $\quad$ **if** $\mathsf{P}'.\mathsf{frq} > \mathcal{H}.\mathsf{top}().\mathsf{frq}$ **then**
6: $\quad\quad \mathsf{P}'.\mathsf{frq} \leftarrow \mathsf{EvaluateMNI}(\mathsf{P}', \mathsf{KG})$ ▷ Algorithm 3
7: $\quad\quad$ **if** $\mathsf{P}'.\mathsf{frq} > \mathcal{H}.\mathsf{top}().\mathsf{frq}$ or $\mathcal{H}.\mathsf{Size} \leq k$ **then**
8: $\quad\quad\quad$ Update $\mathcal{H}$ by $(\mathsf{P}', \mathsf{P}'.\mathsf{frq})$
9: $\quad\quad\quad \mathsf{PatExt}(\mathsf{P}', \mathsf{KG}, \mathcal{CS})$ ▷ Algorithm 2
10: **return** $\mathcal{H}$

Algorithm 2 uses edge-growing to enumerate all possible extensions of a pattern P by adding one edge each time. The new edge to add is determined by an edge-node pair (Line 1), i.e., $(e, v) \in \mathsf{E} \times \mathsf{V}$, in the knowledge graph. It also uses the *DFScode* canonical form of gSpan [1] to avoid generating duplicate extended patterns (Line 2).

Algorithm 2 PatExt(P, KG, $\mathcal{CS}$)

1: **for** each $(e, v) \in \mathsf{E} \times \mathsf{V}$ **do**
2: $\quad$ **if** e can be used to extend P **then** ▷ gSpan
3: $\quad\quad$ Let P' be the extension of P with e
4: $\quad\quad \mathsf{P}'.\mathsf{frq} \leftarrow \mathsf{P}.\mathsf{frq}$
5: $\quad\quad \mathcal{CS} \leftarrow \mathcal{CS} \cup \{(\mathsf{P}', \mathsf{P}'.\mathsf{frq})\}$

Algorithm 3 is the pseudocode of the MNI computation framework in kCP-B, it searches all instances for each node of P (Lines 4 to 9) to produce an exact MNI value.

Algorithm 3 EvaluateMNI(P, KG)

1: $count \leftarrow +\infty$
2: **for** each node $v \in \mathsf{V_P}$ in pattern P **do**
3: $\quad tmp \leftarrow 0$
4: $\quad$ **for** each candidate instance i_v of v **do**
5: $\quad\quad$ **if** i_v has been marked **then**
6: $\quad\quad\quad tmp++$
7: $\quad\quad$ **else if** $\exists$ instance $\mathcal{S}$ of P and $\mathcal{S}$ includes i_v **then**
8: $\quad\quad\quad \mathcal{X}(\mathsf{P}) \leftarrow$ mark $\mathcal{S}$'s nodes to $\mathsf{V_P}$'s domains
9: $\quad\quad\quad tmp++$
10: $\quad$ **if** $tmp < count$ **then** $count \leftarrow tmp$
11: **return** $count$

Algorithm 4 reports the outline of pattern enumeration via meta-index in Section 3.2.1. The algorithm expands an edge only if its source/destination node type is in the out/in-neighbor lists of the node in the current pattern P (Line 2). It also checks the type of the edge and whether adding the new edge-node pair (e, v) will exceed the maximum out- or in- degrees recorded in the meta-index M (Line 3).

Algorithm 4 PatExtMeta(P, M, $\mathcal{CS}$)

1: **for** each node $u \in \mathsf{V_P}$ **do**
2: $\quad$ **for** each $(e, v) \in \mathsf{M}(u).\mathsf{Out} \cup \mathsf{M}(u).\mathsf{In}$ **do**
3: $\quad\quad$ **if** $\mathsf{isValid}(\mathsf{P}, (e, v))$ **then** ▷ node degree checking
4: $\quad\quad\quad$ **if** P can be extended by (e, v) **then** ▷ gSpan
5: $\quad\quad\quad\quad$ Let P' be the extension of P with (e, v)
6: $\quad\quad\quad\quad \mathsf{P}'.\mathsf{ub} \leftarrow \mathsf{P}.\mathsf{frq}$
7: $\quad\quad\quad\quad \mathcal{CS} \leftarrow \mathcal{CS} \cup \{(\mathsf{P}', \mathsf{P}'.\mathsf{ub})\}$

The algorithm below (i.e., Algorithm 5) shows the pseudocode of join$^+$ MNI computation, which is a core subroutine of FastPat framework (see in Section 3.2.4). Algorithm 5 begins by serializing pattern P into a set of path patterns, and then prunes tuples when conducting join for each path pattern (Line 4). Next, it computes the MNI lower bound of the path patterns by applying tuple reduction, and greedily conducts join on the edge tables with other tables (Line 11). After checking all tuples in $\mathsf{T}(A, B)$, we can confirm that the MNI of P is upper bounded by the MNI of $\mathsf{T}(A, B)$. Thus, we can early terminate the join process when P's current MNI upper bound is smaller than k^{th} frequency θ (Line 14).

Algorithm 5 JoinMNI^+(P, PT, θ)

1: Initialize join result table $\mathsf{R} \leftarrow \emptyset$
2: Serialize pattern P into a set of path patterns $\mathcal{PS}$
3: **for** each path pattern $\mathcal{P} \in \mathcal{PS}$ **do**
4: $\quad \mathsf{PT} \leftarrow$ prune tuples from PT by table join on $\mathcal{P}$
5: Apply tuple reduction on each edge table of PT
6: $\mathsf{MNI}_{lb}(\mathsf{P}) \leftarrow$ compute the MNI lower bound of P
7: **while** $\mathcal{PS}$ is not empty **do**
8: $\quad \mathcal{PS} \leftarrow \mathcal{PS} - \{\mathcal{P} \leftarrow \mathsf{argmin}_{P \in \mathcal{PS}} \mathsf{MNI}_{lb}(P)\}$
9: $\quad \mathsf{T}_{\mathcal{P}} \leftarrow$ edge tables of path pattern $\mathcal{P}$ from PT
10: $\quad$ **while** $\mathsf{T}_{\mathcal{P}}$ is not empty **do**
11: $\quad\quad \mathsf{T}_{\mathcal{P}} \leftarrow \mathsf{T}_{\mathcal{P}} - \{\mathsf{T}(A, B) \leftarrow \mathsf{argmin}_{T \in \mathsf{T}_{\mathcal{P}}} \mathsf{MNI}(T)\}$
12: $\quad\quad \mathsf{R} \leftarrow$ join $\mathsf{T}(A, B)$ with $\forall \mathsf{T}_i \in \mathsf{PT} - \{\mathsf{T}(A, B)\}$
13: $\quad\quad \mathsf{P.ub} \leftarrow$ the MNI of verified $\mathsf{T}_{\mathcal{P}}$ in R
14: $\quad\quad$ **if** $\mathsf{P.ub} < \theta$ **then** ▷ Pruning by MNI upper bound
15: $\quad\quad\quad \mathsf{P.frq} \leftarrow \mathsf{P.ub}$ and return
16: $\mathsf{P.frq} \leftarrow \mathsf{MNI}(\mathsf{R})$

Algorithm 6 is the pseudocode of the baseline solution (i.e., FPD-B) for kCPD problem (see in Section 4.1). The basic idea of FPD-B is as follows. First, FPD-B initializes a max-heap $\mathcal{CS}$ for checking the candidate patterns in a best-first manner. For the most frequent pattern P' in $\mathcal{CS}$, it finds the extensions of P' via the edge growing method (Line 5), then evaluates the MNI of each extended pattern in KG (Line 7). Next, it greedily finds the most frequent pattern in $\mathcal{CS}$ and insert the pattern into the result set $\mathcal{R}$ if it can pass diversity check (Lines 9 to 11). Finally, the process

stops when k patterns have been selected or all patterns in $\mathcal{CS}$ have been examined.

Algorithm 6 FPD-B($\mathsf{KG}, \mathsf{P}, k, \lambda$)

```
Initialize max-heap candidate pattern set CS ← {(P, 0)}
Initialize top-k pattern result set R ← ∅
while |R| < k and CS is not empty do
    CS.delete(P' ← CS.top())
    E ← PatExt(P', KG, ∅)                          ▷ Algorithm 2
    for each candidate pattern d ∈ E do
        d.frq ← EvaluateMNI(d, KG)                 ▷ Algorithm 3
        CS ← CS ∪ {(d, d.frq)}
    P'' ← CS.top()
    if |R| = 0 or ∀p ∈ R, Div(p, P'') ≥ λ then
        R ← R ∪ {P''};
return R
```

APPENDIX B
PROOFS

We next give the proofs of lemmas in the paper.

Proof of Lemma 1. For each node $v \in \mathsf{V_P}$, if the node v's instance u is not in $\mathsf{VDom}(v)$, it also cannot be in $\mathsf{F}(v)$, the image set of v holds $\mathsf{F}(v) \subseteq \mathsf{VDom}(v)$. We have:

$$\begin{aligned} &\forall v \in \mathsf{V_P}, \mathsf{F}(v) \subseteq \mathsf{VDom}(v) \\ \Leftrightarrow &\forall v \in \mathsf{V_P}, |\mathsf{F}(v)| \leq |\mathsf{VDom}(v)| \\ \Leftrightarrow &\min_{\forall v \in \mathsf{V_P}} |\mathsf{F}(v)| \leq \min_{\forall v \in \mathsf{V_P}} |\mathsf{VDom}(v)| \end{aligned}$$

Hence, it holds $\mathsf{MNI}(\mathsf{P}) \leq \mathsf{MNI}_{ub}(\mathsf{P})$.

Proof of Lemma 2. We construct table $\mathsf{T}'(A, B)$ as follows: (i) group all tuples in $\mathsf{T}(A, B)$ according to distinct values in $\mathsf{Dom}(\mathsf{T}.A)$ and form $\mathsf{MNI}(\mathsf{T}.A)$ groups, then randomly sample one tuple from each group and insert it into $\mathsf{T}'(A, B)$; (ii) group all tuples in $\mathsf{T}(A, B)$ by the distinct values in $\mathsf{Dom}(\mathsf{T}.B)$ and form $\mathsf{MNI}(\mathsf{T}.B)$ groups, randomly sample one tuple per group and insert it into $\mathsf{T}'(A, B)$ until $\mathsf{MNI}(\mathsf{T}'.B) = \mathsf{MNI}(\mathsf{T}.A)$. We have $\mathsf{MNI}(\mathsf{T}') = \mathsf{MNI}(\mathsf{T}) = \mathsf{MNI}(\mathsf{T}.A)$ as step (i) and (ii) ensure $|\mathsf{Dom}(\mathsf{T}'.A)| = \mathsf{MNI}(\mathsf{T}.A)$ and $|\mathsf{Dom}(\mathsf{T}'.B)| = \mathsf{MNI}(\mathsf{T}.A)$, respectively. The total number of tuples in $\mathsf{T}'(A, B)$ is bounded by $2 * \mathsf{MNI}(\mathsf{T}.A)$ as both step (i) and step (ii) insert at most $\mathsf{MNI}(\mathsf{T}.A)$ tuples.

Proof of Lemma 3. It holds as every tuple that appears in the join result of the reduced tables (i.e., R) is included in the exact join result.

Proof of Lemma 4. First, given a graph $G(V, E)$ and an integer n. The decision version of Maximum Independent Set (MIS) problem asks whether there exists a maximum independent set $I = \{v_1, v_2, \cdots, v_m\}$ of node set V, such that $|I| \geq n$. We denote MIS problem instance as $\langle G(V, E), n\rangle$, this problem is shown to be NP-hard [2].

Second, consider a knowledge graph KG, a core pattern P, a diversity threshold λ and an integer k. Let V_P includes all the patterns extended from core pattern P, i.e., $V_\mathsf{P} = \{\mathsf{P}_1, \mathsf{P}_2, \cdots, \mathsf{P}_m\}$. For the sake of NP reduction, we set $\mathsf{MNI}(\mathsf{P}_i) = 1$ for each candidate pattern P_i in V_P. Then we transform the kCPD problem by creating an edge set E_P. For every two candidate patterns P_i and P_j in V_P, there is an edge e_{ij} in E_P iff $\mathsf{Div}(\mathsf{P}_i, \mathsf{P}_j) < \lambda$. In other words, we connect these similar candidate patterns by an edge, thus, the diversified patterns are not connected. Until now, we obtain a graph $G_\mathsf{P}(V_\mathsf{P}, E_\mathsf{P})$. Thus, the decision version of our simplified and transformed kCPD problem is whether there exists a set S which includes k independent nodes from $G_\mathsf{P}(V_\mathsf{P}, E_\mathsf{P})$. We denote the problem instance as $\langle G_\mathsf{P}(V_\mathsf{P}, E_\mathsf{P}), k\rangle$.

Third, reducing the MIS problem instance $\langle G(V, E), n\rangle$ to our simplified and transformed kCPD problem instance $\langle G_\mathsf{P}(V_\mathsf{P}, E_\mathsf{P}), k\rangle$ is straightforward, i.e., setting $k = n$, and mapping every node in V to V_P and every edge in E to E_P. It costs polynomial time, i.e., $O(|V| + |E|)$. If there exists an independent set I of size k in MIS problem instance $\langle G(V, E), n\rangle$, then there exists a size-k set S for our simplified and transformed kCPD problem instance $\langle G_\mathsf{P}(V_\mathsf{P}, E_\mathsf{P}), k\rangle$. The reason is that every two nodes v_i, v_j in I correspond to two diversified patterns $\mathsf{P}_i, \mathsf{P}_j$ in S with diversity score $\mathsf{Div}(\mathsf{P}_i, \mathsf{P}_j) \geq \lambda$. Conversely, it also holds.

Proof of Lemma 5. We analyze the approximate ratio of the greedy algorithm FPD-B (i.e., Algorithm 6) as follows. First, we denote P^* as the most frequent pattern extended from the core pattern in KG. Let $\mathcal{R}$ and $\mathcal{R}'$ denote the result output by algorithm FPD-B and an exact algorithm, respectively. Next, we require k patterns returned by the exact algorithm, i.e., $|\mathcal{R}'| = k$. Then, there would be two possible cases for the greedy algorithm FPD-B: (i) It identifies pattern sets with the largest cardinality while being smaller than k, i.e., $|\mathcal{R}| < k$. Since $\mathcal{R}$ always includes pattern P^* (i.e., $\mathsf{MNI}(\mathsf{P}^*) = max_{\mathsf{P}_i \in \mathcal{R}}\{\mathsf{MNI}(\mathsf{P}_i)\}$), the approximate ratio:

$$\alpha = \frac{\sum_{\mathsf{P}_i \in \mathcal{R}'} \mathsf{MNI}(\mathsf{P}_i)}{\sum_{\mathsf{P}_j \in \mathcal{R}} \mathsf{MNI}(\mathsf{P}_j)} \leq \frac{k \cdot \mathsf{MNI}(\mathsf{P}^*)}{1 \cdot \mathsf{MNI}(\mathsf{P}^*)} = k.$$

(ii) It identifies the set with k patterns that has the largest total frequency among all pattern sets, i.e., $|\mathcal{R}| = |\mathcal{R}'| = k$, we have the approximate ratio of algorithm FPD-B is

$$\begin{aligned} \alpha &= \frac{\sum_{\mathsf{P}_i \in \mathcal{R}'} \mathsf{MNI}(\mathsf{P}_i)}{\sum_{\mathsf{P}_j \in \mathcal{R}} \mathsf{MNI}(\mathsf{P}_j)} \leq \frac{|\mathcal{R}'| \cdot max_{\mathsf{P}_i \in \mathcal{R}'}\{\mathsf{MNI}(\mathsf{P}_i)\}}{|\mathcal{R}| \cdot min_{\mathsf{P}_j \in \mathcal{R}}\{\mathsf{MNI}(\mathsf{P}_j)\}} \\ &\leq \frac{|\mathcal{R}'|}{|\mathcal{R}|} \cdot \frac{\mathsf{MNI}(\mathsf{P}^*)}{min_{\mathsf{P}_j \in \mathcal{R}}\{\mathsf{MNI}(\mathsf{P}_j)\}} = \frac{max_{\mathsf{P}_i \in \mathcal{R}}\{\mathsf{MNI}(\mathsf{P}_i)\}}{min_{\mathsf{P}_j \in \mathcal{R}}\{\mathsf{MNI}(\mathsf{P}_j)\}}. \end{aligned}$$

Note that both approximation ratios are not constant. The first case shows the approximate ratio for the greedy algorithm can be k in the worst case. However, k is usually not large and the worst case seldomly happened in practice.

Proof of Lemma 6. By noting that the relations $\mathcal{M}_\mathsf{P} \subseteq \mathcal{C}_\mathsf{P} \subseteq \mathcal{D}_\mathsf{P}$, we have

$$\begin{aligned} \mathsf{Div}(\mathsf{P}_1, \mathsf{P}_2) &= 1 - \frac{|\mathcal{C}_{\mathsf{P}_1} \cap \mathcal{C}_{\mathsf{P}_2}|}{|\mathcal{C}_{\mathsf{P}_1} \cup \mathcal{C}_{\mathsf{P}_2}|} \geq 1 - \frac{|\mathcal{C}_{\mathsf{P}_1} \cap \mathcal{D}_{\mathsf{P}_2}|}{|\mathcal{C}_{\mathsf{P}_1} \cup \mathcal{C}_{\mathsf{P}_2}|} \\ &\geq 1 - \frac{|\mathcal{C}_{\mathsf{P}_1} \cap \mathcal{D}_{\mathsf{P}_2}|}{|\mathcal{C}_{\mathsf{P}_1} \cup \mathcal{M}_{\mathsf{P}_2}|} = \mathsf{Div}_{lb}(\mathsf{P}_1, \mathsf{P}_2). \end{aligned}$$

Hence, it holds that $\mathsf{Div}(\mathsf{P}_1, \mathsf{P}_2) \geq \mathsf{Div}_{lb}(\mathsf{P}_1, \mathsf{P}_2)$.

APPENDIX C
COMPLEXITY ANALYSIS

In this part we analyze the time and space complexities of the proposed algorithms in kCP and kCPD.

C.1 Complexity Analysis of kCP Algorithms

Time Complexity of Algorithm kCP-B. The time cost is dominated by two core sub-routines: pattern enumeration (Algorithm 2) and MNI computation (Algorithm 3). Given a knowledge graph $\mathsf{KG}(\mathsf{V},\mathsf{E})$, a core pattern $\mathsf{P}(\mathsf{V_P},\mathsf{E_P})$, the number of all possible candidate pattern is $2^{|\mathsf{E}|}$, as analyzed in [3]. For each pattern, the time complexity to compute its MNI value is $O((|\mathsf{V_P}|+k)\cdot|\mathsf{V}|^{|\mathsf{V_P}|+k-1})$ according to the time cost analysis in GRAMI [4]. Thus, the total time complexity of kCP-B is $O(2^{|\mathsf{E}|}\cdot(|\mathsf{V_P}|+k)\cdot|\mathsf{V}|^{|\mathsf{V_P}|+k-1})$.

Time Complexity of Algorithm kCP-A. It consists of three parts: meta-index and edge table construction, pattern enumeration, upper bound and join-based MNI computation. We analyze the time cost of each part as following. First, the time complexity to construct meta-index and edge table is $O(|\mathsf{V}|+|\mathsf{E}|)$, as it only scans all nodes and edges once. Second, the number of enumerated possible candidate patterns is $(|\mathsf{V_P}|+k)^k d_{max}^k$ according to Algorithm 4. In particular, the largest extension pattern of core pattern P has $(|\mathsf{V_P}|+k)$ nodes with the anti-monotonicity property of MNI. For each pattern P′, the number of generated extensions will less than $(|\mathsf{V_P}|+k)d_{max}$. Third, the time complexity to compute the MNI upper bound is $O(|\mathsf{V}|+|\mathsf{E}|)$ as it only need check all nodes and edges in KG. The time complexity of pattern serialization (Algorithm 5) is $O((|\mathsf{V_P}|+k)+(|\mathsf{E_P}|+k))$ as it only incurs DFS search once. The time complexity join operators in $\mathsf{JoinMNI}^+$ (Algorithm 5) is $O(|\mathsf{E}|^{(|\mathsf{E_P}|+k)})$ as we conduct table join for every edge in the candidate pattern. In summary, the total time complexity of kCP-A is $O(|\mathsf{V}|+|\mathsf{E}|+(|\mathsf{V_P}|+k)^k d_{max}^k\cdot(|\mathsf{V}|+|\mathsf{E}|+(|\mathsf{V_P}|+k)+(|\mathsf{E_P}|+k)+|\mathsf{E}|^{(|\mathsf{E_P}|+k)})$. It is $O((|\mathsf{V_P}|+k)^k d_{max}^k\cdot|\mathsf{E}|^{(|\mathsf{E_P}|+k)})$ after simplification.

According to the theoretical time complexity analysis of kCP-B and kCP-A, our proposed advanced approach kCP-A is significantly better than kCP-B as it reduced a large amount of candidate patterns. In addition, our proposed optimization techniques (e.g., meta-index, edge table, MNI upper bound and tuple reduction in two-pass join scheme) improve the overall performance obviously, as we shown in Section 5.

Space Complexity of Algorithm kCP-B. The space cost of kCP-B is contributed by the MNI computation, which is adapted from GRAMI, we refer the interested reader for the reported space cost in [4].

Space Complexity of Algorithm kCP-A. It is dominated by intermediated result table size of join-based MNI computation. In particular, for each candidate pattern, join-based MNI computation joins at most $(|\mathsf{E_P}|+k)$ edge tables with the anti-monotonicity property of MNI. For each edge table, the maximum table size is $|\mathsf{E}|$, thus, the total space cost is $O(|\mathsf{E}|^{(|\mathsf{E_P}|+k)})$.

C.2 Complexity Analysis of kCPD Algorithms

Time Complexity of Algorithm FPD-B. The time cost is dominated by three core sub-routines: pattern enumeration (Algorithm 2), MNI evaluation (Algorithm 3) and diversity computation. The time complexity of the first two sub-routines are analyzed in Section C.1. In particular, the time cost of pattern enumeration is $O(2^{|\mathsf{E}|})$. For the time cost of MNI computation, the node size of the extension pattern is $|\mathsf{V}|$ in the worst case as the diversity measure does not has anti-monotonicity property. Thus, the time complexity to compute the MNI value of each candidate pattern is $O(|\mathsf{V}|^{|\mathsf{V}|})$, i.e., the cost of adpated GRAMI solution. For each candidate pattern, it costs $O(k\cdot|\mathsf{V}|^2)$ to compute the diversity score among all these candidate patterns in top-k result set $\mathcal{R}$. Hence, the total time complexity of FPD-B is $O(2^{|\mathsf{E}|}\cdot(|\mathsf{V}|^{|\mathsf{V}|}+k\cdot|\mathsf{V}|^2))$.

Time Complexity of Algorithm FPD-A. Since the diversity score function does not have anti-monotonicity property, FPD-A will enumerate all possible candidate patterns, which is the same as FPD-B. Thus, the only difference of the time complexity between FPD-A and FPD-B is the join-based MNI computation. For each pattern, the time cost of join-based MNI computation is $O(|\mathsf{E}|^{|\mathsf{E}|})$ as the maximum number of edges of the candidate pattern is $|\mathsf{E}|$. Hence, the total time complexity of kCP-A is $O(2^{|\mathsf{E}|}\cdot(|\mathsf{E}|^{|\mathsf{E}|}+k\cdot|\mathsf{V}|^2))$ with the diversity score computation cost analyzed above.

Even though the theoretical time complexity of our advanced solution FPD-A is similar with the baseline solution FPD-B, the practical performance of FPD-A is at least one order of magnitude faster than FPD-B due to our suite of optimizations, including meta-index, tuple reduction in join-based method, and our diversity lower bound techniques. The effectiveness of these techniques was evaluated in Section 5.

Space Complexity of Algorithm FPD-A. The space cost of FPD-A is similar with kCP-A but the maximum number of edges in candidate pattern is $|\mathsf{E}|$, where $|\mathsf{E}|$ is the cardinality of edge set in KG. Thus, the space complexity of FPD-A is $O(|\mathsf{E}|^{|\mathsf{E}|})$.

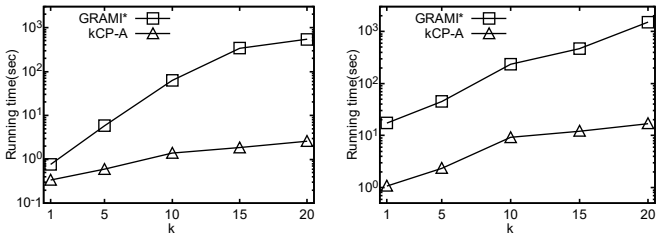
APPENDIX D
COMPLEMENTARY EXPERIMENT RESULTS

We report complementary experiment results for kCP and kCPD problems as follows.

D.1 Performance Evaluation on kCP Problem

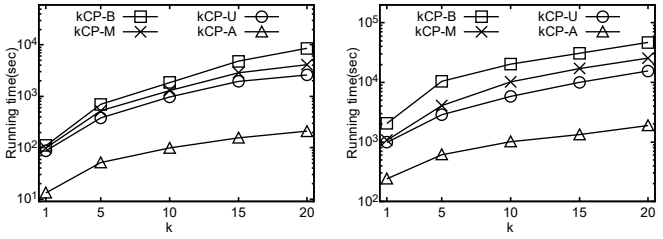
Effect of k. (1) We adapted the state-of-the-art method GRAMI [4] to find top-k frequent patterns of the whole knowledge graph KG, denotes as GRAMI*. It is a baseline of our kCP problem with input core pattern $\mathsf{P}=\emptyset$. For efficiency study, we revised kCP-A for finding top-k frequent patterns without a core, then compared it with GRAMI*. Figures 1(a) and (b) show the running time of kCP-A and GRAMI* on our default datasets by varying k. kCP-A achieves up to 210.2x and 89.5x faster than GRAMI* for Oscar and COVID-19, respectively. The experimental results confirm the superiority of kCP-A over its state-of-the-art method GRAMI*. (2) We present more efficiency results for large graphs, i.e., MiCo and YAGO3. Figure 2(a) reveals that the main performance gain comes from our join-based MNI computation instead of meta-index. In addition, the performance gap between kCP-A and its competitors (i.e., kCP-B, kCP-M and kCP-U) widens when k increases for all datasets, e.g., it achieves 24.8x on YAGO3, which suggests that kCP-A has good scalability on large datasets.

Memory Usage. We next present the memory usage of our algorithms. To measure the size of Java objects, we



(a) Oscar (b) COVID-19

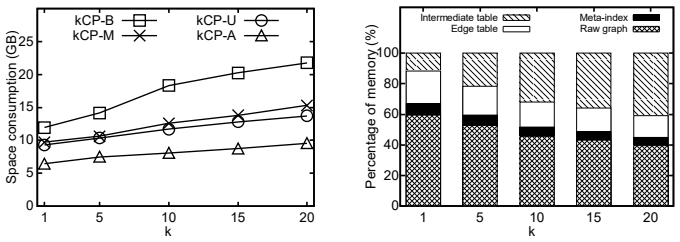
Fig. 1. Varying k: discover top-k frequent patterns without core pattern



(a) MiCo (b) YAGO3

Fig. 2. Effect of k, kCP problem

used a third-party tool *RamUsageEstimator* from Apache Lucene[1] in our implementation. Each reported memory is the peak memory that totally consumed by the algorithm. We calculate the memory usage (by counting the size of every constructed data structures) of the four solutions for kCP on the largest dataset YAGO3. As shown in Figure 3(a), the space consumptions of all methods are increasing with the number of frequent patterns k. kCP-A has the lowest memory usage among all methods as it employs the two-pass join scheme, which only materializes some of the pattern instances. In Figure 3 (b), we analyzed the space consumptions of different components in kCP-A by varying k. The intermediate table size of kCP-A increases with k as kCP-A needs to compute MNI for more candidate patterns at large k.



(a) Total space consumption (b) Space consumption of kCP-A

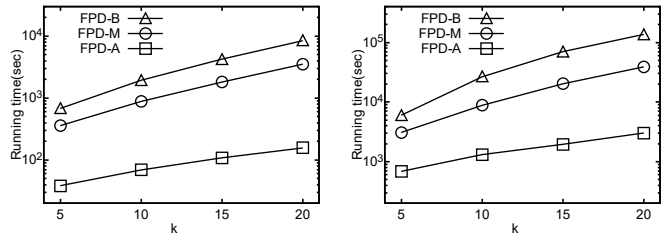
Fig. 3. Varying k: memory usage of kCP solutions, YAGO3

D.2 Performance Evaluation on kCPD Problem

Effect of k. We briefly present the experimental results of kCPD algorithms by varying k over large datasets. As shown in Figure 4, FPD-A performs the best among all the solutions, it achieves a speedup over one order of magnitude against FPD-B for MiCo and the largest dataset (i.e., YAGO3), which suggests that FPD-A has good scalability.

D.3 Case Study for kCPD Problem

We visualized the top-5 result patterns of GRAMI* (i.e., the top-5 most frequent patterns returned by GRAMI [4] without inputting a core), kCP (top-5 frequent patterns) and kCPD algorithms (top-5 frequent and diversified patterns) with core pattern P : Actor $\xrightarrow{\text{character is}}$ Character $\xrightarrow{\text{from}}$ Film, λ = 0.3) on the Oscar dataset in Figure 5. (I) the result patterns returned by GRAMI* are trivial. For example, 80% of the top-5 patterns returned by GRAMI* only includes one edge, however, the average edge size of the top-5 patterns produced by kCP and kCPD are 3 and 4. (II) kCPD discovers top-5 patterns with diversified node/edge types, e.g., its overlapping ratio (i.e., the average percentage of each node type exists in top-k patterns) is 12%, while the corresponding ratio on kCP and GRAMI* is 17% and 44%, respectively. (III) Compared to kCP problem, kCPD returns patterns that not only cover common entity types, e.g., "Actor" and "Award", but also involve uncommon entity types from the film crew, e.g., "Cinematographer". This provides more diversified options for Oscar knowledge graph exploration.



(a) MiCo (b) YAGO3

Fig. 4. Effect of k, kCPD problem

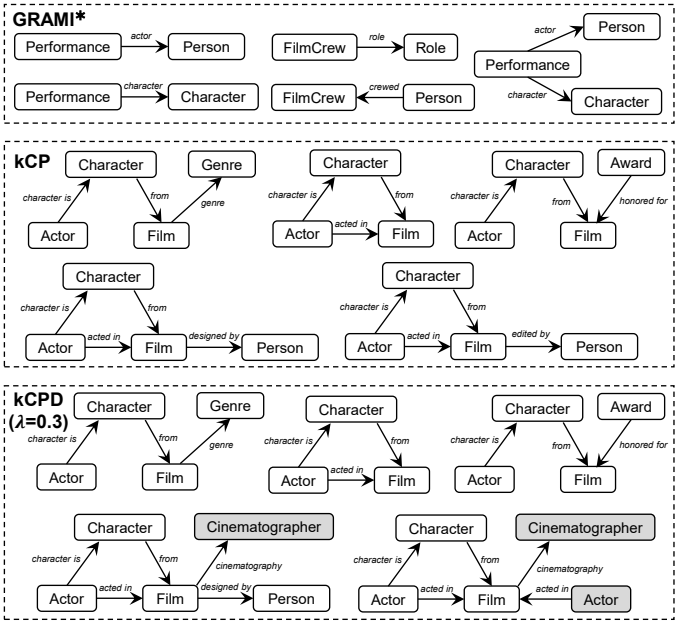


Fig. 5. Top-5 patterns returned by the three problems, Oscar

REFERENCES

[1] X. Yan and J. Han, "gspan: Graph-based substructure pattern mining," in *ICDM*, 2002, pp. 721–724.
[2] M. Kuramochi and G. Karypis, "Finding frequent patterns in a large sparse graph," *DMKD*, vol. 11, no. 3, pp. 243–271, 2005.
[3] A. Prateek, A. Khan, A. Goyal, and S. Ranu, "Mining top-k pairs of correlated subgraphs in a large network." *PVLDB*, vol. 13, no. 9, pp. 1511–1524, 2020.
[4] M. Elseidy, E. Abdelhamid, S. Skiadopoulos, and P. Kalnis, "Grami: Frequent subgraph and pattern mining in a single large graph," *PVLDB*, vol. 7, no. 7, pp. 517–528, 2014.

1. https://lucene.apache.org/